UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P O Box 1450
Alexandria, Virginia 22313-1450
www uspto gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/803,663	03/18/2004	Jimmie Earl DeWitt JR.	AUS920030548US1	7005

35525 7590 05/02/2008

IBM CORP (YA)
C/O YEE & ASSOCIATES PC
P.O. BOX 802333
DALLAS, TX 75380

EXAMINER	
WANG, BEN C	
ART UNIT	**PAPER NUMBER**
2192	

NOTIFICATION DATE	DELIVERY MODE
05/02/2008	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

ptonotifs@yeeiplaw.com

Office Action Summary	Application No.	Applicant(s)	
	10/803,663	DEWITT ET AL.	
	Examiner	Art Unit	
	BEN C. WANG	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *17 January 2008*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-11,13,15-18,20,21 and 23* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☐ Claim(s) *1-11, 13, 15-18, 20-21, and 23* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application

6) ☐ Other: _____.

DETAILED ACTION

1. Applicant's amendment dated January 17, 2008, responding to the Office action mailed October 18, 2007 provided in the rejection of claims 1-13, 15-18, 20-21, and 23, wherein claims 1, 18, 21, and 23 are amended, claims 12 is canceled.

Claims 1-11, 13, 15-18, 20-21, and 23 remain pending in the application and which have been fully considered by the examiner.

Applicant's arguments with respect to claims currently amended have been fully considered but are moot in view of the new grounds of rejection – see Grunwald *et al.* - art of record (IDS- 8/24/07), as applied hereto.

Claim Rejections – 35 USC § 103(a)

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

2. Claims 1-11, 13, 15-18, 20-21, and 23 are rejected under 35 U.S.C. 103(a) as being unpatentable over by Baba et al. (Pub. No. US 2002/0010733 A1) (hereinafter 'Baba') in view of Hussain et al. (Pat. No. US 6,658,416 B1) (hereinafter 'Hussain') and

further in view of Grunwald et al. (*Whole-Program Optimization for Time and Space Efficient Threads, 1996 ACM*) (hereinafter 'Grunwald' - - art of record)

3. **As to claim 1** (Currently Amended), Baba discloses a method in a data processing system for autonomically determining execution flow of a computer program, comprising:

- providing a set of hardware registers for identifying a work area for a thread of the computer program, wherein the work area stores thread tracking information for the thread (e.g., [0029] – Any registers in the processor may be assigned as <u>the control register group</u>. The stack machine rewrites <u>the control register group</u> as needed as it accesses <u>the work area</u> for <u>the current thread</u> according to the data in the register group; in this way it executes <u>thread</u>);

Baba does not explicitly disclose retrieving symbolic data for the thread, wherein retrieving symbolic data for the tread includes retrieving symbolic data from an indexed symbolic database by searching the indexed symbolic database for symbolic data based on a process identifier for the thread; generating a call sequence of the computer program based on the symbolic data for the thread.

However, in an analogous art of *Apparatus and Method for Creating an Indexed database of Symbolic Data for Use with Trace Data of a Computer Program*, Hussain discloses:

- retrieving symbolic data for the thread, wherein retrieving symbolic data for the tread includes retrieving symbolic data from an indexed symbolic

database by searching the indexed symbolic database for symbolic data based on a process identifier for the thread (e.g., Fig. 10A, element of 1005 – Symbolic Data; Fig. 10B, element of 1070 – Symbolic Data; Fig. 13B, element of 1340; Fig. 18, elements 1830 – Search Indexed Database for Symbol Data Matching PID and Address, 1850 – Display Symbolic Data In Accordance With Trace File; Col. 16, Line 30 – Process identification (pid)); and

- generating a call sequence of the computer program based on the symbolic data for the thread (e.g., Col. 2, Lines 38-41 – Such designers employ <u>profiling tools</u> to find characteristic <u>code sequences</u> and/or single instructions that require optimization for the available software for a given type of hardware).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hussain into the Baba's system to further provide retrieving symbolic data for the thread, wherein retrieving symbolic data for the tread includes retrieving symbolic data from an indexed symbolic database by searching the indexed symbolic database for symbolic data based on a process identifier for the thread; generating a call sequence of the computer program based on the symbolic data for the thread in Baba system.

The motivation is that it would further enhance the Baba's system by taking, advancing and/or incorporating Hussain's system which offers significant advantages that store symbolic data for loaded modules during or shortly after a performance trace and utilizes the stored symbolic data when performing a performance analysis at later

time; the merged symbol file contains information useful in performing symbolic resolution of address information in trace files for each instance of a module as once suggested by Hussain (e.g., Abstract)

Furthermore, Baba discloses using the set of hardware registers (e.g., [0029] – Any registers in the processor may be assigned as the control register group. The stack machine rewrites the control register group as needed as it accesses the work area for the current thread according to the data in the register group; in this way it executes thread), but Baba and Hussain do not explicitly disclose determining whether an overflow is about to occur in the work area; responsive to determining that an overflow is about to occur in the work area, copying the thread tracking information from the work area to a buffer.

However, in an analogous art of *Whole-Program Optimization for Time and Space Efficient Threads*, Grunwald discloses determining whether an overflow is about to occur in the work area (e.g., Sec. 2 Heap Allocation of Stacks for Threaded Applications, 2nd Par., Lines 4-6 - … rely on the MMU hardware to detect overflow); responsive to determining that an overflow is about to occur in the work area, copying the thread tracking information from the work area to a buffer (e.g., Sec. 2.1 Combined Heap Allocation, 1st Par., Lines 4-9 - … to allocate new stack space when there is a potential for overflow …).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Grunwald into the Baba-Hussain's system to further provide determining whether an overflow is about to occur

in the work area; responsive to determining that an overflow is about to occur in the work area, copying the thread tracking information from the work area to a buffer using the set of hardware registers in the Baba-Hussain system.

The motivation is that it would further enhance the Baba-Hussain's system by taking, advancing and/or incorporating the Grunwald's system which offers significant advantages to reduce time and space thread overhead using control flow and register liveness information inferred after compilation as once suggested by Grunwald (e.g., Abstract, 2nd Par.)

4. **As to claim 2** (Original) (incorporating the rejection in claim 1), Baba discloses the method wherein the set of hardware registers includes a work area register, a work area length register, and a current pointer register (e.g., [0029] – Any registers in the processor may be assigned as <u>the control register group</u>. The stack machine rewrites <u>the control register group</u> as needed as it accesses <u>the work area</u> for <u>the current thread</u> according to the data in the register group; in this way it executes <u>thread</u>; Fig. 4 – element 22 – <u>Register for Stack Head Pointer</u>; Fig. 6 – element 33 – <u>Work Area</u> for Switching Thread; P. 4, Left-Col., Lines 51-56 – A sidetracking component to sidetrack a current register data indicating the current state data of execution of a program stored in a control register group in said stack machine in response to said switching request to switch threads, and stored said current register data of execution in a sidetracking area setup in a current thread).

5. **As to claim 3** (Original) (incorporating the rejection in claim 2), Baba discloses the method wherein the work area register includes a pointer pointing to a beginning of the work area for the thread (e.g., [0029] – Any registers in the processor may be assigned as the control register group. The stack machine rewrites the control register group as needed as it accesses the work area for the current thread according to the data in the register group; in this way it executes thread; Fig. 4 – element 22 – Register for Stack Head Pointer; Fig. 6 – element 33 – Work Area for Switching Thread; P. 4, Left-Col., Lines 51-56 – A sidetracking component to sidetrack a current register data indicating the current state data of execution of a program stored in a control register group in said stack machine in response to said switching request to switch threads, and stored said current register data of execution in a sidetracking area setup in a current thread).

6. **As to claim 4** (Original) (incorporating the rejection in claim 2), Baba discloses the method wherein the work area length register includes one of a size of the work area for the thread or a pointer pointing to an end of the work area for the thread (e.g., [0029] – Any registers in the processor may be assigned as the control register group. The stack machine rewrites the control register group as needed as it accesses the work area for the current thread according to the data in the register group; in this way it executes thread; Fig. 4 – element 22 – Register for Stack Head Pointer; Fig. 6 – element 33 – Work Area for Switching Thread; P. 4, Left-Col., Lines 51-56 – A sidetracking component to sidetrack a current register data indicating the current state

data of execution of a program stored in a control register group in said stack machine in response to said switching request to switch threads, and stored said current register data of execution in a sidetracking area setup in a current thread).

7. **As to claim 5** (Previously Presented) (incorporating the rejection in claim 2), Baba discloses the method wherein the current pointer register includes a pointer pointing to a location of the work area where last thread tracking information is written (e.g., [0029] – Any registers in the processor may be assigned as <u>the control register group</u>. The stack machine rewrites <u>the control register group</u> as needed as it accesses <u>the work area</u> for <u>the current thread</u> according to the data in the register group; in this way it executes <u>thread</u>; P. 4, Left-Col., Lines 51-56 – A sidetracking component to sidetrack a current register data indicating the current state data of execution of a program stored in a control register group in said stack machine in response to said switching request to switch threads, and stored said current register data of execution in a sidetracking area setup in a current thread).

8. **As to claim 6** (Original) (incorporating the rejection in claim 1), Hussain discloses the method wherein the thread tracking information for the thread includes a plurality of call stack entries for the thread (e.g., Col. 9, Lines 22 –35 – Java® stack are used to store the state of Java® method invocations; When <u>a thread invokes a method</u>, the JVM <u>pushes</u> a new frame onto the <u>Java stack</u> of the thread. When <u>the method completes</u>, the JVM <u>pops the frame</u> for that method and discards it).

9. **As to claim 7** (Original) (incorporating the rejection in claim 6), Hussain discloses the method wherein each of the plurality of call stack entries is written upon detection of one of a method call and a method return for the thread (e.g., Col. 9, Lines 22 –35 – Java® stack are used to store the state of Java® method invocations; When a thread invokes a method, the JVM pushes a new frame onto the Java stack of the thread. When the method completes, the JVM pops the frame for that method and discards it).

10. **As to claim 8** (Original) (incorporating the rejection in claim 7), Hussain discloses the method wherein each of the plurality of call stack entries includes an address to and an address from which one of the method call and a method return is executed (e.g., Col. 9, Lines 22 –35 – Java® stack are used to store the state of Java® method invocations; When a thread invokes a method, the JVM pushes a new frame onto the Java stack of the thread. When the method completes, the JVM pops the frame for that method and discards it).

11. **As to claim 9** (Original) (incorporating the rejection in claim 8), Hussain discloses the method wherein each of the plurality of call stack entries further includes additional information, and wherein the additional information includes time stamps and performance monitoring counter values (e.g., Col. 9, Lines 22 –35 – Java® stack are used to store the state of Java® method invocations; When a thread invokes a method, the JVM pushes a new frame onto the Java stack of the thread. When the method

completes, the JVM pops the frame for that method and discards it; Col. 1, Lines 65-67 – Typically, a time-stamped record, where 'time' is defined as any monitonically increasing metric, such as, number of instructions executed, is produced for each such event; Col. 2, Lines 20-22 – at each interruption, information is recorded for a predetermined length of time or for a predetermined number of events of interest)

12. **As to claim 10** (Original) (incorporating the rejection in claim 9), Hussain discloses the method wherein the additional information is compressed with the address to and the address from which one of the method call and a method return is executed when each of the plurality of call stack entries is written (e.g., Col. 9, Lines 22 –35 – Java® stack are used to store the state of Java® method invocations; When a thread invokes a method, the JVM pushes a new frame onto the Java stack of the thread. When the method completes, the JVM pops the frame for that method and discards it).

13. **As to claim 11** (Previously Presented) (incorporating the rejection in claim 9), Hussain discloses the method wherein the additional information is compressed with the address to and the address from which one of the method call and a method return is executed when the thread tracking information is copied from the work area to a buffer (e.g., Col. 9, Lines 22 –35 – Java® stack are used to store the state of Java® method invocations; When a thread invokes a method, the JVM pushes a new frame onto the Java stack of the thread. When the method completes, the JVM pops the frame for that method and discards it; Fig. 11 – an exemplary diagram of performance trace data that

may be stored as a trace file or maintained in the trace buffer; Fig. 15 – a flowchart outlining an exemplary operation of the present invention when generating an indexed database of symbolic data from performance trace data stored in the trace buffer in a dynamic manner).

14. **As to claim 13** (Previously Presented) (incorporating the rejection in claim 1), Hussain discloses the method wherein the buffer is one of a trace buffer and a consolidated buffer accessible by an application (e.g., Col. 10, Lines 42-47 – These trace hooks are employed to send trace data to trace program, which stores the trace data in buffer. The trace data in buffer may be subsequently stored in a file for post-processing, or the trace data may be processed in real time; Fig. 4 – element 404 – Buffer; Fig. 6 – Generate Merge File; Fig. 9 – element 910 – Merge File).

15. **As to claim 15** (Previously Presented) (incorporating the rejection in claim 1), Hussain discloses the method wherein the symbolic data matches the process identifier for the thread and the address of one of a method call and a method return for the thread (e.g., Fig. 18, steps 1810, 1820, and 1830; Col. 21, Line 58 through Col. 22, Line 2 – As shown in Fig. 18, the operation starts with reading the trace file (step 1810). The process identifier (pid) and address information are obtained form the trace file (step 1820)).

16. **As to claim 16** (Previously Presented) (incorporating the rejection in claim 1), Hussain discloses the method wherein retrieving symbolic data for the thread includes retrieving symbolic data from one of a directory of the loaded module, a shadow directory, or a loaded module if an address of the loaded module on a disk is known (e.g., Fig. 17, element 1710, 1720; Col. 15, Lines 27-56; Col. 16, Lines 5-18 – the present invention allows for generating symbols out of a different directory than the one from which the system loads the modules; Col. 21, Lines 14-22; Col. 22, Lines 25-34, 41-49).

17. **As to claim 17** (Previously Presented) (incorporating the rejection in claim 1), Hussain discloses the method wherein generating a call sequence of the computer program includes associating the retrieved symbolic data with the thread tracking information in the buffer (e.g., Fig. 8 – steps 801 – Obtain Interrupted Thread ID, 804 – Identify Method Block Being Interrupted, 806 – Send Trace Information; Col. 9, Lines 22-24 – Java stacks are used to store <u>the state of Java method invocations</u>. When a new thread is launched, the JVM creates a new Java stack for <u>the thread</u>; Fig. 10A – Symbolic Data – Symbolic Name, Offset, Length; Fig. 10B – Symbolic Data (0), Symbolic Data (1), Symbolic Data (2); Fig. 13B – Step – Identify Symbolic Data Using Offset; Fig. 18 – Step 1820 – <u>Obtain Process ID (PID)</u> and <u>Address of</u> <u>Desired Symbol</u>).

18. **As to claim 18** (Currently Amended), Baba discloses a data processing system for autonomically determining execution flow of a computer program, the data processing system comprising:

- providing means for providing a set of hardware registers for identifying a work area for a thread of the computer program, wherein the work area stores thread tracking information for the thread (e.g., [0029] – Any registers in the processor may be assigned as <u>the control register group</u>. The stack machine rewrites <u>the control register group</u> as needed as it accesses <u>the work area</u> for <u>the current thread</u> according to the data in the register group; in this way it executes <u>thread</u>);

Baba does not explicitly disclose retrieving means for retrieving symbolic data from the thread, wherein the retrieving means comprises searching means for searching an indexed symbolic database for symbolic data based on a process identifier for the thread; and generating means for generating a call sequence of the computer-program based on the symbolic data for the thread.

However, in an analogous art of *Apparatus and Method for Creating an Indexed database of Symbolic Data for Use with Trace Data of a Computer Program*, Hussain discloses:

- retrieving means for retrieving symbolic data from the thread, wherein the retrieving means comprises searching means for searching an indexed symbolic database for symbolic data based on a process identifier for the thread (e.g., Fig. 10A, element of 1005 – Symbolic Data; Fig. 10B, element of 1070 – Symbolic Data; Fig. 13B, element of 1340; Fig. 18, elements 1830 –

Search Indexed Database for Symbol Data Matching PID and Address, 1850 – Display Symbolic Data In Accordance With Trace File; Col. 16, Line 30 – Process identification (pid)); and

- generating means for generating a call sequence of the computer-program based on the symbolic data for the thread (e.g., Col. 2, Lines 38-41 – Such designers employ profiling tools to find characteristic code sequences and/or single instructions that require optimization for the available software for a given type of hardware).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hussain into the Baba's system to further provide retrieving means for retrieving symbolic data from the thread, wherein the retrieving means comprises searching means for searching an indexed symbolic database for symbolic data based on a process identifier for the thread; and generating means for generating a call sequence of the computer-program based on the symbolic data for the thread in Baba system.

The motivation is that it would further enhance the Baba's system by taking, advancing and/or incorporating Hussain's system which offers significant advantages that store symbolic data for loaded modules during or shortly after a performance trace and utilizes the stored symbolic data when performing a performance analysis at later time; the merged symbol file contains information useful in performing symbolic resolution of address information in trace files for each instance of a module as once suggested by Hussain (e.g., Abstract)

Furthermore, Baba discloses using the set of hardware registers (e.g., [0029] – Any registers in the processor may be assigned as the control register group. The stack machine rewrites the control register group as needed as it accesses the work area for the current thread according to the data in the register group; in this way it executes thread), but Baba and Hussain do not explicitly disclose determining means for determining whether an overflow is about to occur in the work area; coping means, responsive to determining that an overflow is about to occur in the work area, copying the thread tracking information from the work area to a buffer.

However, in an analogous art of *Whole-Program Optimization for Time and Space Efficient Threads*, Grunwald discloses determining means for determining whether an overflow is about to occur in the work area (e.g., Sec. 2 Heap Allocation of Stacks for Threaded Applications, 2nd Par., Lines 4-6 - … rely on the MMU hardware to detect overflow); coping means, responsive to determining that an overflow is about to occur in the work area, copying the thread tracking information from the work area to a buffer (e.g., Sec. 2.1 Combined Heap Allocation, 1st Par., Lines 4-9 - … to allocate new stack space when there is a potential for overflow …).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Grunwald into the Baba-Hussain's system to further provide determining means for determining whether an overflow is about to occur in the work area; coping means, responsive to determining that an overflow is about to occur in the work area, copying the thread tracking information from the work area to a buffer in the Baba-Hussain system.

The motivation is that it would further enhance the Baba-Hussain's system by taking, advancing and/or incorporating the Grunwald's system which offers significant advantages to reduce time and space thread overhead using control flow and register liveness information inferred after compilation as once suggested by Grunwald (e.g., Abstract, 2^{nd} Par.)

19. **As to claim 20** (Previously Presented) (incorporating the rejection in claim 18), Hussain discloses the data processing system wherein the generating means comprises: associating means for associating the retrieved symbolic data with the thread tracking information in the buffer (e.g., Fig. 8 – steps 801 – Obtain Interrupted Thread ID, 804 – Identify Method Block Being Interrupted, 806 – Send Trace Information; Col. 9, Lines 22-24 – Java stacks are used to store the state of Java method invocations. When a new thread is launched, the JVM creates a new Java stack for the thread; Fig. 10A – Symbolic Data – Symbolic Name, Offset, Length; Fig. 10B – Symbolic Data (0), Symbolic Data (1), Symbolic Data (2); Fig. 13B – Step – Identify Symbolic Data Using Offset; Fig. 18 – Step 1820 – Obtain Process ID (PID) and Address of Desired Symbol).

20. **As to claim 21** (Currently Amended), Baba discloses a computer program product, including a computer recordable-type medium storing computer readable program code for determining execution flow of a computer program, the computer program product comprising:

- first instructions for providing a set of hardware registers for identifying a work area for a thread of the computer program, wherein the work area stores thread tracking information for the thread (e.g., [0029] – Any registers in the processor may be assigned as <u>the control register group</u>. The stack machine rewrites <u>the control register group</u> as needed as it accesses <u>the work area</u> for <u>the current thread</u> according to the data in the register group; in this way it executes <u>thread</u>);

Baba does not explicitly disclose third instructions for retrieving symbolic data for the thread, wherein the third instructions comprises sub-instructions for searching an indexed symbolic database for symbolic data based on a process identifier for the thread; fourth instructions for generating a call sequence of the computer program based on the symbolic data for the thread.

However, in an analogous art of *Apparatus and Method for Creating an Indexed database of Symbolic Data for Use with Trace Data of a Computer Program*, Hussain discloses:

- third instructions for retrieving symbolic data for the thread, wherein the third instructions comprises sub-instructions for searching an indexed symbolic database for symbolic data based on a process identifier for the thread (e.g., Fig. 10A, element of 1005 – Symbolic Data; Fig. 10B, element of 1070 – Symbolic Data; Fig. 13B, element of 1340; Fig. 18, elements 1830 – Search Indexed Database for Symbol Data Matching PID and Address, 1850 – Display Symbolic Data In Accordance With Trace File; Col. 16, Line 30 – Process identification (pid));

- fourth instructions for generating a call sequence of the computer program based on the symbolic data for the thread (e.g., Col. 2, Lines 38-41 – Such designers employ profiling tools to find characteristic code sequences and/or single instructions that require optimization for the available software for a given type of hardware).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hussain into the Baba's system to further provide third instructions for retrieving symbolic data for the thread, wherein the third instructions comprises sub-instructions for searching an indexed symbolic database for symbolic data based on a process identifier for the thread; fourth instructions for generating a call sequence of the computer program based on the symbolic data for the thread in Baba system.

The motivation is that it would further enhance the Baba's system by taking, advancing and/or incorporating Hussain's system which offers significant advantages that store symbolic data for loaded modules during or shortly after a performance trace and utilizes the stored symbolic data when performing a performance analysis at later time; the merged symbol file contains information useful in performing symbolic resolution of address information in trace files for each instance of a module as once suggested by Hussain (e.g., Abstract)

Furthermore, Baba discloses using the set of hardware registers (e.g., [0029] – Any registers in the processor may be assigned as the control register group. The stack machine rewrites the control register group as needed as it accesses the work area for

the current thread according to the data in the register group; in this way it executes thread), but Baba and Hussain do not explicitly disclose second instructions for determining whether an overflow is about to occur in the work area; third instruction, responsive to determining that an overflow is about to occur in the work area, for copying the thread tracking information from the work area to a buffer.

However, in an analogous art of *Whole-Program Optimization for Time and Space Efficient Threads*, Grunwald discloses second instructions for determining whether an overflow is about to occur in the work area (e.g., Sec. 2 Heap Allocation of Stacks for Threaded Applications, 2nd Par., Lines 4-6 - … rely on the MMU hardware to detect overflow); third instruction, responsive to determining that an overflow is about to occur in the work area, for copying the thread tracking information from the work area to a buffer (e.g., Sec. 2.1 Combined Heap Allocation, 1st Par., Lines 4-9 - … to allocate new stack space when there is a potential for overflow …).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Grunwald into the Baba-Hussain's system to further provide second instructions for determining whether an overflow is about to occur in the work area; third instruction, responsive to determining that an overflow is about to occur in the work area, for copying the thread tracking information from the work area to a buffer using the set of hardware registers in the Baba-Hussain system.

The motivation is that it would further enhance the Baba-Hussain's system by taking, advancing and/or incorporating the Grunwald's system which offers significant

advantages to reduce time and space thread overhead using control flow and register liveness information inferred after compilation as once suggested by Grunwald (e.g., Abstract, 2nd Par.)

21. **As to claim 23** (Currently Amended) (incorporating the rejection in claim 21), Hussain discloses the computer program product wherein the fourth instruction comprises: sub-instructions for associating the retrieved symbolic data with the thread tracking information in the buffer (e.g., Fig. 8 – steps 801 – Obtain Interrupted Thread ID, 804 – Identify Method Block Being Interrupted, 806 – Send Trace Information; Col. 9, Lines 22-24 – Java stacks are used to store the state of Java method invocations. When a new thread is launched, the JVM creates a new Java stack for the thread; Fig. 10A – Symbolic Data – Symbolic Name, Offset, Length; Fig. 10B – Symbolic Data (0), Symbolic Data (1), Symbolic Data (2); Fig. 13B – Step – Identify Symbolic Data Using Offset; Fig. 18 – Step 1820 – Obtain Process ID (PID) and Address of Desired Symbol).

Conclusion

22. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ben C. Wang whose telephone number is 571-270-1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Ben C Wang/
Examiner, Art Unit 2192
April 24, 2008

/Tuan Q. Dam/

Supervisory Patent Examiner, Art Unit 2192